The Evolution of Randomess Testing

Martin Ukrop, Petr Švenda, Vashek Matyáš

xukrop,svenda,matyas@fi.muni.cz

Faculty of Informatics
Masaryk University
Brno, Czech Republic

Abstract

We propose a novel, automated method of creating statistical randomness tests. Tests are created as hardware-like circuits using EACirc, the framework for automatic problem solving based on genetic programming. The basic overview of the framework is given along with the summary of used settings and a set of reference experiments. The framework is then used to assess the randomness of outputs produced by chosen eStream cipher candidates and SHA-3 hash function candidates. Success of the tests generated by EACirc is compared to standard statistical batteries (Sts Nist, Dieharder). Results of one chosen case are analysed in detail.

Keywords: statistical randomness, random distinguisher, evolutionary algorithms, genetic programming, software circuit, SHA-3, eStream.

1 Introduction

Producing random data by computers is extremely difficult[1], as they are inherently deterministic. Yet the quality of random data is crucial for many cryptographic applications. To ease the testing of randomness, batteries of statistical randomness tests such as Sts Nist [Rukhin et al., 2010] or Dieharder [Brown, 2004] have been developed. Although convenient in some ways, statistical randomness testing based on human-designed tests has several important drawbacks. The test creation must be preceded by an idea of mathematical property and its thorough analysis, which can be extremely time- and people-consuming. Further on, the tests are limited to one particular property and testing other properties requires beginning the testing process all over again. Both of the above-mentioned problems would be resolved if tests of comparable quality could be generated automatically, without the help of human specialists.

In this paper, a novel method of non-randomness testing is given. The proposed approach is based on evolutionary algorithms and utilizes the idea of software-emulated circuits. Its main benefits lie in easy automation and high potential of creating new tests, thus surpassing the disadvantages of statistical batteries. This novel method is implemented in a general problem-solving framework called EACirc.

The following sections use the EACirc framework in practical tests and compare all obtained results with already existing approach using statistical batteries. The experiments are divided into three categories: control experiments checking the sanity of our implementation and used referential data, experiments examining randomness of stream cipher outputs and experiments assessing randomness of hash function outputs. A single case is then chosen for a more detailed analysis.

Evolutionary algorithms were already used to probe specific problems of a particular function (e.g., DES, TEA, XTEA) in the past, yet for their really useful application, the identification of specific sub-problems like deviation of χ^2 Goodness of Fit tests applied to statistic of least significant bits [Castro and Viñuela, 2005] was required. Our approach may directly provide a distinguisher without prior identification of such sub-problems as well as be used when such a property was identified, providing a higher degree of freedom when searching for a distinguisher.

2 Previous work

Numerous works tackled the problem of distinguisher construction between data produced by cryptographic functions and truly random data, both with reduced and full number of rounds. Usually, statis-

[1] *"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."* [von Neumann, 1951]

tical testing with battery of tests (e.g., STS NIST or Dieharder) or additional custom tailored statistical tests are performed.

TEA algorithm [Wheeler and Needham, 1995] with a reduced number of rounds is a frequent target for cryptoanalysis with genetic algorithms. In [Castro and Viñuela, 2005], a successful randomness distinguisher for XTEA limited to 4 rounds is generated with genetic algorithms. The distinguisher generates a bit mask with high Hamming weight which when applied to function input, resulting in deviated χ^2 Goodness of Fit test of the output. Subsequent work [Hu, 2010] improves an earlier attack with quantum-inspired genetic algorithms, finding more efficient distinguishers for a reduced round TEA algorithm and succeeding for the 5-round TEA.

In [Turan et al., 2006], detailed examination of eStream Phase 2 candidates (full and reduced round tests) with STS NIST battery and structural randomness tests was performed, finding six ciphers deviating from expected values. More recently, the same battery, but only a subset of the tests, was applied to the SHA-3 candidates (in the second round of competition, 14 in total) for a reduced number of rounds as well as only to the compression function of algorithm [Doganaksoy et al., 2010]. 256-bit versions of SHA-3 finalists were subjected to statistical tests using a GPU-accelerated evaluation [Kaminsky, 2012]. Both algorithms and selected tests from STS NIST battery were implemented for the nVidia CUDA platform. Because of massive parallelization, superpoly tests introduced by [Dinur and Shamir, 2009] were possible to be performed, detecting some deviations in all but the Grøstl algorithm.

An important difference of our approach from previous work is the production of a program (in the form of a software circuit) that provides different results depending on given inputs. Previous work produced a fixed result, e.g., bit mask in [Castro and Viñuela, 2005, Hu, 2010] that is directly applied to all inputs.

To some extent, the structure of a software circuit resembles artificial neural networks (deep belief neural networks in particular [Hinton et al., 2006]). Notable differences are in the learning mechanism and circuit dimensions (neural networks usually use very small number of layers). The function of individual nodes is different as well, since all nodes in artificial neural networks usually perform the same function.

3 Evolution-based randomness testing

In this section we try to describe a method of automatically generating statistical randomness tests. Compared to the standard (manual) way of their creation, our approach has a couple of advantages:

- no prior knowledge of statistical properties of random data is needed;
- test creation does not require excessive human analytical labour;
- tests are dynamically adapting to the testing data;
- atypical and/or yet unknown data properties may be used.

The main idea is to use supervised learning techniques based on evolutionary algorithms. We apply the principles of genetic programming to design and optimize a successful distinguisher [Englund et al., 2007] (a test able to tell the explored data apart from a truly random data). The distinguisher will be represented as a hardware-like circuit consisting of a number of interconnected simple functions.

3.1 Using software-emulated circuits

We consider distinguishers in the form of hardware-like circuits with gates (*function nodes*) and a set of wires (*node connectors*). Each node is responsible for the computation of a simple function on its inputs (e. g. binary AND operation). Function nodes are positioned into layers, where outputs from one layer are connected to inputs of the next. Input of the whole circuit is used as an input for the first layer and output of the last layer is considered the output of the entire circuit. Connectors may only link adjacent layers, but may cross each other (contrary to real single-layer hardware circuits). An example of such hardware-like circuit can be seen in figure 1.

In the current implementation, we consider only simple node functions operating on bytes. These include the standard bit-manipulating functions (AND, ROTL, ...), relation operators (EQ, LT, ...), constant function (CONS) and an empty function (NOP).

It would be sufficient to restrict ourselves to a smaller set of functions (e. g. NAND only), since with such subset we can express arbitrarily complex function. However, the spacial requirements rise with the function complexity. More complex functions in nodes enable us to limit the circuit to significantly smaller number of layers and nodes, while retaining a comparable expressive power. We decided to support a wider variety of functions as a size and human understandability trade-off.

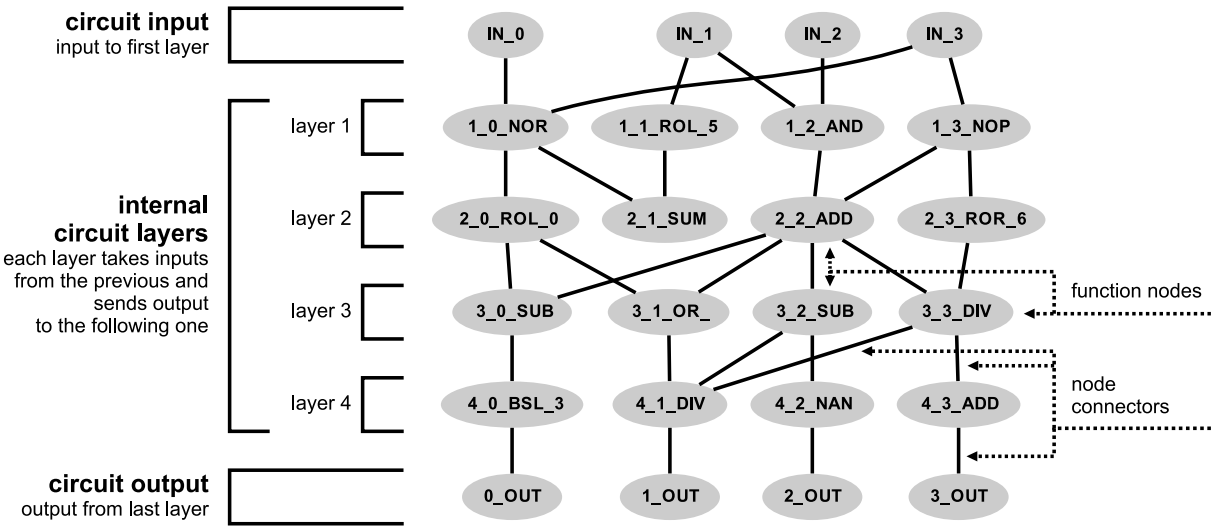

Figure 1: Simple example of software-emulated circuit.

3.2 The EACirc framework

Combining the principles of genetic programming and software circuits, we developed EACirc, the framework for automatic problem solving. The framework tries to evolve a circuit solving the given problem. The process consists of the following steps:

- Firstly, several software circuits are randomly initialized (randomly selected functions in nodes, randomly assigned existence of connectors between nodes) forming population of candidate individuals. Every individual is represented by one circuit. Note that such a random circuit will most probably not provide any meaningful output for given inputs and can even have disconnected layers (no output at all).
- Secondly, we generate a new set of teaching data (if necessary). This data is used as circuit inputs in the evaluation phase.
- Every individual (circuit) is evaluated on every test input from the current set. Based on the outputs, a fitness value (quality measure) is assigned to each circuit.
- Based on the evaluation provided by the fitness function, a potentially improved population is generated by mutation and crossover operators from individuals taken from the previous generation. Design of every individual (circuit) may be changed by changing operations computed in nodes or adding/removing connectors between nodes in subsequent layers.

The whole process (except for the initialization) is repeated multiple times, usually until the desired success rate of the population is achieved or the required number of generations have evolved. A simplified work-flow of the EACirc evolution process is summarized in figure 2.

The initial version of EACirc was created by Petr Švenda at the Laboratory of Security and Applied Cryptography, Masaryk University. Later on, the application was improved by Matej Prišťák and Ondrej Dubovec (as their master and bachelor theses, respectively [Prišťák, 2012, Dubovec, 2012]).

The core evolutionary features are provided by *GAlib*, a C++ Library of Genetic Algorithm Components developed at MIT [Wall, 1995] parametrized by function callbacks (e. g. function for mutation, sexual crossover, fitness function, ...). The framework now contains three different experiments (eStream ciphers, SHA-3 functions and a simple file comparator) with modular structure allowing for effortless integration of new ones.

The fitness function is computed within separate evaluator modules. There are multiple approaches to evaluation, several of which are implemented in EACirc. While most of the previous runs distinguished circuit outputs by taking the topmost bit, we now consider a more universal approach based on output byte categories. The output byte distributions of random and assessed streams are compared using Pearson's χ^2 test.

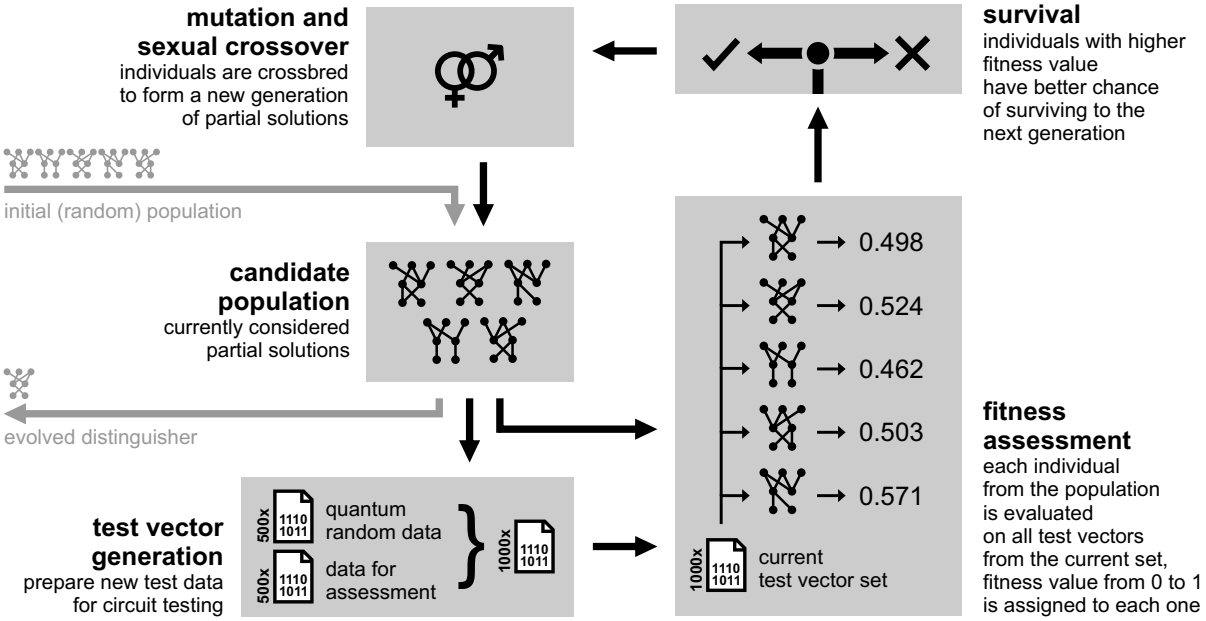

Figure 2: Simplified work-flow of the evolution process in EACirc.

EACirc has a variety of other functions improving the core features of evolutionary algorithms and software circuit emulation. For full details, user and development documentation, see EACirc wiki at *GitHub* [Švenda et al., 2012]. The most important features are the following:

- **Bit-reproducibility**
  EACirc uses genetic programming, which is fundamentally a randomized algorithm. However, to enable experiment replication and results verification, we use controlled pseudorandomness. This allowed us to replicate experiments by just providing the same input files and a fixed central seed.
- **Computation recommencing**
  By saving and loading its entire internal state to a set of XML files, EACirc allows for compuration recommencing. This feature is especially useful for computation-expensive experiments – when the machine is rebooted, we can continue from last saved state instead of starting all over again.
- **Multi-format output**
  For easy reusing and analysis, the evolved circuits are output in 4 different formats:
  - binary output (useful for reloading the circuits into EACirc),
  - graph DOT output (serves as a visual aid to human analyst),
  - simple text output (application-independent export format) and
  - program output (a stand-alone C program used for independent static analysis).

  To further ease the manual human analysis, circuits can be *pruned* before export – all disconnected and unused nodes are removed.
- **CUDA support**
  EACirc supports nVidia CUDA for circuit evaluation during the computation of fitness. When executed on GPU instead of CPU, the evaluation runtime decreases by the coefficient of about 70.

4 Control experiments

Before performing the experiments themselves, we needed to acquire reference results – what does it mean streams are indistinguishable from random in out context? Is the referential random data indistinguishable from random? What is the achieved fitness value for such distinguishers?

4.1 Experiment settings and output data

Most of the general settings (evolution and circuit parameters) were taken from Matej Prišťák's thesis [Prišťák, 2012]. The evolution works with a population of 20 individuals, with a sexual crossover probability of 20 % and a mutation probability set to 5 %. In each case (if not stated otherwise), we evolve

30 000 generations with a test vector set (learning data) changing every 100^{th} generation. Thus, a total of 300 unique test vector sets are used in each run. The circuit dimensions are limited to 5 layers with a maximum of 8 function nodes per layer. It processes up to 16 input bytes and produces 2 output bytes.

Each testing set consists of 1 000 independent vectors, exactly half of which is truly random with the other half is generated from the assessed data. According to the previous research, the imbalance in test vector types complicates the learning phase, since the circuits are also trying to learn which type is more frequent in the particular set. The order of random and non-random vectors in the set is not fixed. Hence (equation 1), all the results output by EACirc are based on a sample of about 2.3 MB of assessed data.

$$\Sigma = \frac{30000 \text{ generations}}{100 \frac{\text{generations}}{\text{test set}}} \cdot \frac{1}{2} \cdot 1000 \frac{\text{vector}}{\text{test set}} \cdot 16 \frac{\text{bytes}}{\text{vector}} \approx 2.29\,\text{MB} \tag{1}$$

The expected circuit output is always `0x00` (zero byte) for a non-random vector and `0xff` (full byte) for a random one. The used evaluator considers each of the output bytes separately, taking bytes with numerical interpretation lower than 128 as indicating a non-random stream and bytes higher than 127 as indicating a random stream. Hence, the decision is based only on the first bit of each output byte. Using the output of the evaluator, the fitness of the circuit is quantified as a quotient of a number of correctly predicted vectors and a total number of vectors in a set.

All the experiments were run 30-times in parallel. This precaution was taken as to mitigate the results fluctuations common in randomized algorithms. The final result presented is the average of these 30 executions. Further discussion and more detailed settings can be found in [Ukrop, 2013].

When using Sts Nist and Dieharder, we used an external file with 250 MB of the assessed stream produced by EACirc. STS NIST was run on 100 sub-streams, each consisting of 1 000 000 bits. This amounts to about 11.92 MB of assessed data. All 15 available test were run in all supported configurations. From the Dieharder suite, only the tests corresponding to the original Diehard collection were used. Each of the chosen tests was rune just once, but was let to process as much data as it required. Running the whole set processed about 582 MB altogether with the smallest test consuming about 3 MB and the largest one about 127 MB.

4.2 Searching for non-randomness in quantum random data

The first control experiment tries to distinguish quantum random data from other quantum random data. We use 193 MB of data obtained from Quantum Random Bit Generator Service [Zagreb, 2007]. We presume to fail at this and thus establish the randomness of the assessed data stream.

Using the standard statistical batteries confirmed out expectations – all used Dieharder and Sts Nist tests failed to provide evidence of the data non-randomness. Running EACirc with the settings described in subsection 4.1 yielded the average maximum fitness value in generations just after the change of test vectors of 0.52 with runs differing in 3^{rd} or 4^{th} decimal place.

We anticipated that the difference of obtained fitness from the naïve value of 0.50 was influenced by population size (bigger population increases fitness variance) and the size of test set (the probability of just guessing correctly decreases with a bigger test set). We performed further experiments and confirmed both these presumptions. We can thus conclude that in our settings the fitness value of 0.52 corresponds to indistinguishable streams.

4.3 Comparing different QRNGs

Secondly, we want to compare quantum random data streams obtained from two different sources ([Zagreb, 2007] and [Berlin, 2010]). We prepared 6 independent files of 5 MB from each source and attempted to find a distinguisher for each pair.

All the results oscillate closely around 0.52 indicating indistinguishable streams (see subsection 4.2). We can thus conclude that, for our purposes, both sources are equally random and equally reliable. Since no of the tested files expressed any statistically significant deviation from the others, we can use these files interchangeably.

4.4 Analysing uncompressed audio streams

The third and last of the control experiments compares the set of audio files. We considered a set of 12 files – 3 quantum random data files, 3 uncompressed audio files with white, pink and Brownian noise, the same noise files with intermediate mp3 compression and 3 samples of uncompressed black-metal music.

The quantum random data files had about 5 MB and were turned into a listenable file by adding a WAV header instructing to interpret the data as 2-channel, 16 bit/sample, 44.1 kHz PCM-encoded audio. The next subset consisted of 30 seconds samples (about 5.3 MB) of white, pink and Brownian noise in the same audio format. The third subset was created from the above-mentioned generated noises by mp3 compression (bitrate of 128 kbps) and decompression back to the PCM-encoded audio. Note that after the lossy MP3 compression the files took about 480 kB each (compared to 5.3 MB of the uncompressed version). The last three were 30 seconds samples of transcendental khaoblack metal by Abbey ov Thelema [Abbey ov Thelema, 2012] all taken from the band's promo called *MMXII: Here & Now - At the Threshold ov End Times*.

Similarly as in subsection 4.3, we attempted to evolve a distinguisher for each pair of the files. We analysed the results in the similar manner and concluded the following:

- generated white noise is completely indistinguishable from random data files,
- pink and Brownian noise are easily told apart from each other or the quantum random files (success rate generally over 80 %),
- mp3 compression has small, but detectable effect on the sound (although nearly undetectable by unskilled human ear, it successfully shifts the distinguisher success rate to about 0.58 when comparing with an uncompressed noise of the same kind),
- used metal samples can be reliably distinguished from white noise (general success over 80 %), less so from pink and Brownian noise (success rate only around 65 %),
- used metal samples are nearly indistinguishable from each other on the binary level (although the differences are easily detectable by human ear).

5 Application to stream ciphers and hash functions

The main motivation for this work is to provide a tool with the crucial ability to automatically probe for unwanted properties of cryptographic functions that signalize flaws in the function design. Therefore, inspired by [Prišťák, 2012], we decided to analyse randomness of stream cipher outputs. We only considered stream ciphers from the recent eStream competition [eStream, 2005], since we could use the unified cipher interface prescribed in the competition.

Similar experiments were performed on candidate hash functions from SHA-3 competition [NIST, 2007] (inspired by [Dubovec, 2012]). As in eStream ciphers, we utilized the unified hash function interface prescribed in the competition. We analysed the randomness of streams produced as a concatenation of hash digests.

5.1 Processing eStream candidates outputs

From 34 candidates in the eStream competition, 23 were potentially usable for testing (due to renamed or updated versions, problems with compilation, ...). Out of these, we limited ourselves to only 7 (Decim, Grain, FUBUKI, Hermes, LEX, Salsa20 and TSC), since these had internal structure that allowed for a simple reduction of complexity by reducing a number of internal rounds. For all used ciphers, the implementation from the last successful phase of the competition was taken. The ciphers were tested in unlimited versions and then for all lower number of rounds until reaching indistinguishability from a random stream.

The experiment results are summarized in table 1. We differentiate outcomes: either both EACirc and statistical batteries were able to distinguish the produced stream from the truly random data, only statistical batteries were able to make the distinction or no approach succeeded. The results indicate that in this case, EACirc performs more or less the same as standard statistical batteries (Decim being the most prominent exception). Dieharder sometimes performed better than STS NIST, but it has to be taken into consideration that it is newer and made decision based on a much larger data sample. In general, both statistical batteries processed longer stream than EACirc (for detailed numbers see subsection4.1). Regarding the matters of speed, EACirc had a comparably longer learning phase, but usually provided a distinguisher working working faster than statistical batteries.

cipher	number of rounds														
	1	2	3	4	5	6	7	8	9	10	11	12	13	15	32
Decim															
FUBUKI															
Grain															
Hermes															
LEX															
Salsa20															
TSC-4															

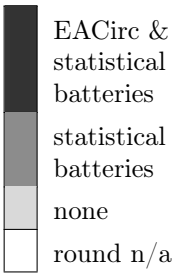

Table 1: Comparing EACirc with statistical testing batteries Dieharder and Sts Nist on eStream cipher candidates outputs.

5.2 Processing SHA-3 candidates outputs

From 64 hash functions that entered the competition, 51 were selected to the first round. Out of these, 42 were potentially usable for testing (due to source code size, speed and compilation problems). The implementations (taken from the last successful phase of the competition) and modifications limiting the number of rounds in these functions were taken over from previous work [Dubovec, 2012] and revised. In the end, 18 most promising candidates were chosen: ARIRANG, Aurora, Blake, Cheetah, CubeHash, DCH, Dynamic SHA, Dynamic SHA2, ECHO, Grøstl, Hamsi, JH, Lesamnta, Luffa, MD6, SIMD, Tangle, and Twister. These were the candidates fulfilling the following two requirements:

- the hash functions could be effortlessly limited in complexity by decreasing the number of internal rounds and
- while the unlimited version produced a random-looking output, their most limited version did not.

As opposed to the work we were inspired by, we generated continuous output stream by hashing a simple 4-byte counter starting from a randomly generated value. We obtained a 256-bit digest, which we cut in half to produce 2 independent test vector inputs of 16 bytes each. In case of generating a continuous stream (for the purposes of Dieharder and Sts Nist), we concatenated the digests.

The results, summarized in table 2, indicate that in this case, EACirc performs slightly worse than standard statistical batteries. Although in most of the cases it found a statistically significant variation from a neutral success rate of 0.52, it can be seen that it often failed in the last round successfully distinguished by statistical batteries. Once again, when interpreting these results, we must be aware of the imbalance of test data available to statistical batteries and EACirc (for detailed numbers see subsection 4.1).

Another observation worth noting is the consistency of the results within the 30 runs of the same experiment. Previously (mainly subsection 5.1), all the results within an experiment were consistent (all 30 runs reached more or less the same results). The computations in this experiment display the variations characteristic to evolutionary algorithms – only some of the runs are successful (the randomized evolution in the other just did not succeed). It may be interesting to consider a larger amount of runs in border cases, where statistical batteries were successful but EACirc was not.

6 Inspection of Salsa20 output stream

After performing a wide range of experiments, we analysed one selected case in a more detailed manner. We studied the dependence of distinguisher success rate on the number of generations already computed. Further attention was paid to the evolved circuit and the statistical properties it uses to draw the final verdict (random vs. non-random).

6.1 Distinguisher success rate

The general relationship between fitness value and the number of evolved generations in evolutionary algorithms is very specific – a typical example can be seen in figure 3. This jaw-like curve represents the the success rate of a circuit trying to distinguish two independent random streams. The success rate

cipher	number of rounds																	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	22	23	24	max
ARIRANG																		
Aurora																		17
Blake																		14
Cheetah																		16
CubeHash																		
DCH																		
DSHA																		16
DSHA2																		17
Echo																		
Grøstl																		
Hamsi																		
JH																		42
Lesamnta																		32
Luffa																		
MD6																		104
Twister																		
SIMD																		
Tangle																		80

distinguished by both EACirc and statistical batteries
distinguished only by statistical batteries
not distinguished
round not available

Table 2: Comparing EACirc with statistical testing batteries Dieharder and Sts Nist on SHA-3 hash candidates outputs.

rises, during the period when the test vector set remains unchanged (100 generations in our setting) and then suddenly drops after the set change. This is caused by the circuit *over-learning* on a specific test vector set (circuits are learning to distinguish this particular set instead of general characteristics of the streams). As can be easily seen in the graph, the behaviour repeats almost periodically. However, notice that the success rate does never exceed the value of 0.55.

The phenomenon of over-learning can be easily suppressed by changing the test vectors more frequently or increasing the number of vectors in a set. On the other hand, higher test set change frequency or more vectors would increase computational complexity. Therefore a reasonable trade-off is used.

In figure 4 we see similar relationship for circuit distinguishing Salsa20 cipher limited to 2 rounds. The over-learning tendency (repeating continual rise and sudden drop) is partly present as well, but in contrast to the previous case the circuits success rate reaches much higher values. Even if not evolving a universal distinguisher, this would be a sufficient evidence for non-randomness of Salsa20 output stream.

We can further notice that after initial fluctuations the circuit success rate show another periodic behaviour about every 4 000 generations. The circuit stabilises at distinguishing the Salsa20 output and then suddenly drops back to about a success of random guessing. It than gets better again and after about 4 000 generations (equivalent to about 450 KB of data) drops again. This behaviour is specific to Salsa20 and its source probably comes the the cipher design. A detailed analysis will be the part of our future work.

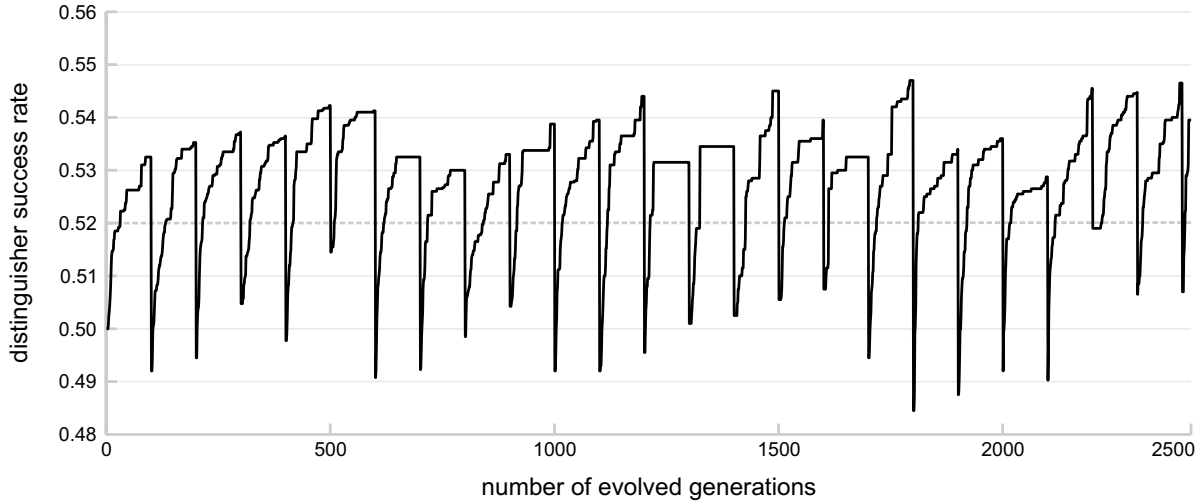

Figure 3: Circuit success rate for control experiment trying to distinguish two streams of quantum random data (note the shifted scale on y-axis). The dotted line represents the value of 0.52 (stream indistinguishable from random).

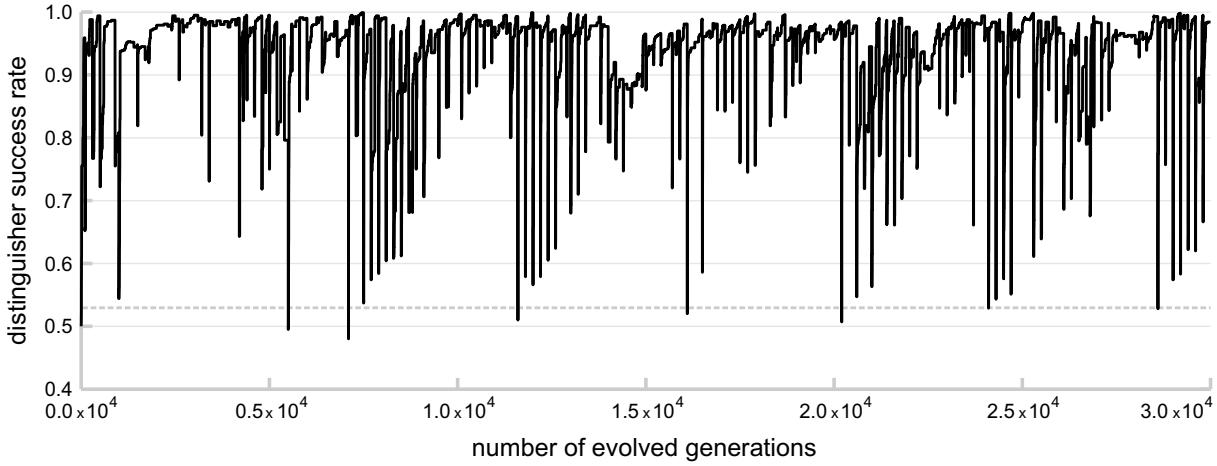

Figure 4: Circuit success rate for distinguishing Salsa20 limited to 2 rounds from quantum random data (note the shifted scale on y-axis). The dotted line represents the value of 0.52 (stream indistinguishable from random).

6.2 Evolved circuit analysis

Other type of detailed study of Salsa20 limited to 2 rounds included the evolved distinguishers. An example interpretation of one such circuit is demonstrated on the circuit shown in figure 5. We took an evolved distinguisher circuit, pruned it (removing all nodes not participating in computing the final fitness), generated 1 000 000 random input sequences for the circuit and inspected the distribution of values coming from every node.

As can be seen in the diagram, we can conclude that both output bytes (1_OUT and 2_OUT) depend solely on 7^{th} input byte (IN_6). Circuits evolved in parallel runs exhibited very similar behaviour – in many of them, the output bytes (and thus the final verdict) depended only on the 7^{th} input byte. It is difficult to tell what is the exact form of this weakness, but it draws out attention to the ever-mentioned byte 7. It definitely implies a possible design flaw in Salsa20 limited to 2 rounds influencing the randomness of every 7^{th} output byte.

After analysing the output of internal nodes, we concentrated on the circuit output and its final verdict (random vs. non-random). We used circuit from figure 5, where both output bytes produce the same value (this happened in most of the parallel runs). We made statistics of all values output by the circuit

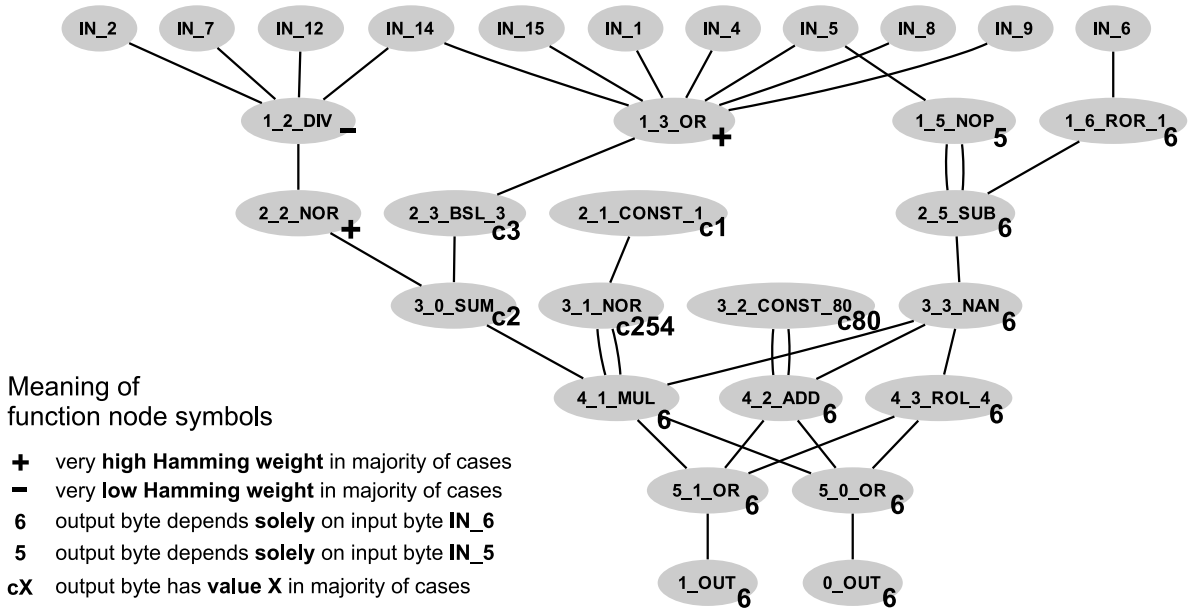

Figure 5: Analysis of a distinguisher evolved for Salsa20 limited to 2 rounds (pruned version of the circuit is displayed).

for random data stream and Salsa20 (limited to 2 rounds) output stream. In both cases, 1 000 000 input sequences were used. The frequencies of output values are plotted in figure 6 in form of a cumulative bar chart. Each coloured block represents a single value, its width proportional to the number of times this value was output.

The evaluator used interpreted the circuit outputs as follows:

- if the numerical value of the output byte was less or equal to 127, the stream was concluded to originate from Salsa20;
- otherwise (numerical value of the output byte equal to or greater than 128) it was concluded to be from a random source.

Two important conclusions can be drawn from the data in figure 6. Firstly, the circuit succeeds in distinguishing Salsa20 limited to 2 rounds most of the times, but not always. Secondly, while the distribution of output values in case of random stream is more or less even, in case of Salsa20 the value of 126 was far more frequent (85.02%). From the latter fact, we could possibly backtrack through the circuit to establish the exact bits in the 7^{th} input byte causing the stream to be distinguishable from random, if such analysis was required.

7 Conclusions

This work explored automated methods of creating statistical randomness tests. Tests were created as hardware-like circuits using EACirc, framework for automatic problem solving based on genetic programming principles.

Firstly, capabilities of the framework were checked by numerous reference experiments. The assumed behaviour when trying to distinguish two sets of quantum random data (even from different sources) was confirmed. A set of audio files was confronted with various types of noise and random data.

After performing these control experiments, cryptographically interesting applications of randomness were investigated. The randomness of 7 different eStream cipher outputs was assessed. The evaluation was done both using the proposed automated method (EACirc) and utilising standard statistical batteries (Dieharder, Sts Nist) and the results were compared. An analogical set of experiments was performed on 18 SHA-3 candidate hash functions. EACirc results for Salsa20 were thoroughly analysed, demonstrating the usage of information provided by EACirc.

Based on our experience and the experimental results obtained, we can draw several conclusions concerning the proposed method of automatic randomness test generation:

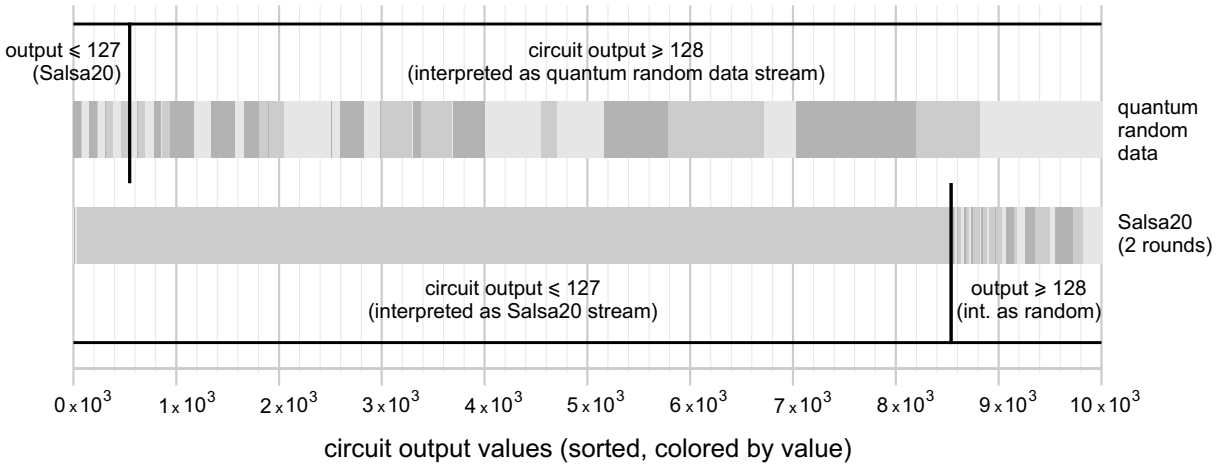

Figure 6: Frequency of circuit output values for the circuit displayed in 5 for 2 streams. Evaluator interprets values less than 128 as stream originating from the cipher, values above or equal to 128 as stream originating from random source.

- **Success rate**
  From the point of success rate, the proposed method is generally comparable to the standard sets of statistical batteries. Results sometimes differ in border cases, in favour of statistical batteries. These border cases should be the goal of further research as the differences may be caused by improper settings and/or insufficient computation time. The difference may also lie in unequal input sample lengths (see below).
- **Amount of data used**
  In general, smaller data samples were provided to EACirc (at most, we used about 2.5 MB) than to statistical batteries (about 12 MB in case of Sts Nist and more than 200 MB in case of Dieharder). Note that some test may provide indication of failure even when less data is available.
- **Atypical approach**
  The proposed method uses a significantly different approach to detect non-randomness compared to statistical batteries. It does not require prior knowledge of specific data properties – instead, it tries to deduce these properties by itself. Therefore, possibilities of using yet unknown data properties arises. This, however, was not conclusively proven, since we have done a wide analysis instead of concentrating on the best possible performance for a particular function.
- **Limited input scope**
  Since the distinguisher circuits only process small parts of the input at a time, this approach may by unable to detect non-randomness present in the global scale. Enabling the circuit to process longer inputs would alleviate this drawback.
- **Speed and complexity**
  The proposed evolution-based approach has a very slow (and computation-intensive) learning phase compared to the use of statistical batteries. Nevertheless, when a working distinguisher is found, assessing further data is very fast.
- **Dynamically adapting distinguishers**
  While tests from standard statistical batteries look for a predefined evidence of non-randomness, distinguishers evolved by EACirc dynamically adapt to the data stream. Thus, if a data stream changes its properties, the test will evolve accordingly (predefined statistical tests never change).
- **Results interpretation**
  On one hand, dynamically adapting tests present a huge disadvantage when interpreting their results – it may be very difficult for humans to analyse, on what data properties is the distinguisher basing its verdict. On the other hand, statistical tests only inform of the data's global characteristics (e. g. there are much more ones than zeroes), while the distinguisher circuits may be a little more specific (e. g. every 4^{th} byte has a higher Hamming weight than it should).

7.1 Proposed future work

As could be seen in section 5, EACirc still falls behind standard statistical batteries in some cases. In the future work we will concentrate on those border cases were EACirc is outperformed.

Another primary goal for us will be enabling the circuit to process longer inputs and thus detect more global interdependencies. We consider enabling the circuit to read further input byte by using special in-node READ function. Other method would be to implement a kind of *memory* for the circuit, which would enable the transfer of information when processing longer inputs.

Another interesting idea is to explore the range of functions allowed in the circuit nodes. On one hand, we may allow more complex data processing in a single node – sequences extracted from the byte-code of the analysed stream cipher/hash function may be used. On the other hand, we may limit the range of allowed functions to but a few, e. g. only AND, OR and NOT, as such a small set is sufficient to express arbitrarily complex function if given enough space.

Furthermore, we plan to perform deeper analysis of the obtained results with respect to the tested stream ciphers and hash functions. For this, new tools for interpreting the results will have to be developed (e. g. statistical analyser of the node outputs in evolved circuits).

7.2 Acknowledgments

This work was supported by the GAP202/11/0422 project of the Czech Science Foundation. The access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005) is highly appreciated.

References

[Abbey ov Thelema, 2012] Abbey ov Thelema (2012). MMXII: Here & Now - At the Threshold ov End Times. `http://bandzone.cz/abbeyovthelema` Accessed: 2013-05-04.

[Berlin, 2010] Berlin (2010). High bit rate quantum random number generator service. Humboldt University of Berlin. `http://qrng.physik.hu-berlin.de/` Accessed: 2013-05-03.

[Brown, 2004] Brown, R. G. (2004). Dieharder: A random number test suite. Duke University Physics Department. `http://www.phy.duke.edu/~rgb/General/dieharder.php` Accessed: 2013-05-03.

[Castro and Viñuela, 2005] Castro, J. C. H. and Viñuela, P. I. (2005). New results on the genetic cryptanalysis of TEA and reduced-round versions of XTEA. *New Gen. Comput.*, 23(3):233–243.

[Dinur and Shamir, 2009] Dinur, I. and Shamir, A. (2009). Cube attacks on tweakable black box polynomials. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 278–299. Springer-Verlag.

[Doganaksoy et al., 2010] Doganaksoy, A., Ege, B., Koçak, O., and Sulak, F. (2010). Statistical analysis of reduced round compression functions of SHA-3 second round candidates. Technical report, Institute of Applied Mathematics, Middle East Technical University, Turkey.

[Dubovec, 2012] Dubovec, O. (2012). Automated search for dependencies in SHA-3 hash function candidates. Bachelor thesis, Faculty of Informatics Masaryk University. `http://is.muni.cz/th/324866/fi_b_a2/` Accessed: 2013-05-04.

[Englund et al., 2007] Englund, H., Hell, M., and Johansson, T. (2007). A note on distinguishing attacks. In *Information Theory for Wireless Networks, 2007 IEEE Information Theory Workshop on*, pages 1–4. IEEE.

[eStream, 2005] eStream (2005). Call for Stream Cipher Primitives. `http://www.ecrypt.eu.org/stream/call/` Accessed: 2013-05-04.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[Hu, 2010] Hu, W. (2010). Cryptanalysis of TEA using quantum-inspired genetic algorithms. *Journal of Software Engineering and Applications*, 3(1):50–57.

[Kaminsky, 2012] Kaminsky, A. (2012). GPU parallel statistical and cube test analysis of the SHA-3 finalist candidate hash functions. In *15th SIAM Conference on Parallel Processing for Scientific Computing (PP12)*.

[NIST, 2007] NIST (2007). SHA-3: Cryptographic hash algorithm competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html` Accessed: 2013-05-04.

[Prišťák, 2012] Prišťák, M. (2012). Automated search for dependencies in eStream stream ciphers. Master thesis, Faculty of Informatics Masaryk University. `http://is.muni.cz/th/172546/fi_m/` Accessed: 2013-05-04.

[Rukhin et al., 2010] Rukhin, A. et al. (2010). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. `http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf` Accessed 2013-05-03.

[Turan et al., 2006] Turan, M. S., Doğanaksoy, A., and Ç. Çalik (2006). Detailed statistical analysis of synchronous stream ciphers. In *ECRYPT Workshop on the State of the Art of Stream Ciphers (SASC'06)*.

[Ukrop, 2013] Ukrop, M. (2013). Usage of evolvable circuit for statistical testing of randomness. Bachelor thesis, Faculty of Informatics Masaryk University. `http://is.muni.cz/th/374297/fi_b/` Accessed: 2013-09-29.

[von Neumann, 1951] von Neumann, J. (1951). Various techniques used in connection with random digits. *Applied Mathematics Series*, 12:36–38.

[Wall, 1995] Wall, M. (1995). GAlib: A C++ Library of Genetic Algorithm Components. `http://lancet.mit.edu/ga/` Accessed: 2013-05-04.

[Wheeler and Needham, 1995] Wheeler, D. and Needham, R. (1995). TEA, a tiny encryption algorithm. In *Fast Software Encryption*, pages 363–366. Springer.

[Zagreb, 2007] Zagreb (2007). Quantum Random Bit Generator Service. Ruđer Bošković Institute, Zagreb. `http://random.irb.hr/index.php` Accessed: 2013-05-03.

[Švenda et al., 2012] Švenda, P., Ukrop, M., Prišťák, M., et al. (2012). Eacirc: Framework for autmatic search for problem solving circuit via evolutionary algorithms. Laboratory of Security and Applied Cryptography, Masaryk University. `http://github.com/petrs/EACirc` Accessed: 2013-05-04.